SOFTWARE INSTALLATION GUIDE

Extreme vSLX Installation and User Guide, 18r.1.00a

Emulating the ExtremeRouting SLX 9850 and ExtremeSwitching SLX 9540 Devices

9035472-01
November 2018

Legal Notice

Extreme Networks, Inc. reserves the right to make changes in specifications and other information contained in this document and its website without prior notice. The reader should in all cases consult representatives of Extreme Networks to determine whether any such changes have been made.

The hardware, firmware, software or any specifications described or referred to in this document are subject to change without notice.

Trademarks

Extreme Networks and the Extreme Networks logo are trademarks or registered trademarks of Extreme Networks, Inc. in the United States and/or other countries.

All other names (including any product names) mentioned in this document are the property of their respective owners and may be trademarks or registered trademarks of their respective companies/owners.

For additional information on Extreme Networks trademarks, please see: www.extremenetworks.com/company/legal/trademarks

Software Licensing

Some software files have been licensed under certain open source or third-party licenses. End-user license agreements and open source declarations can be found at: www.extremenetworks.com/support/policies/software-licensing

Support

For product support, phone the Global Technical Assistance Center (GTAC) at 1-800-998-2408 (toll-free in U.S. and Canada) or +1-408-579-2826. For the support phone number in other countries, visit: http://www.extremenetworks.com/support/contact/

For product documentation online, visit: https://www.extremenetworks.com/documentation/

Contents

Preface **7**
- Conventions 7
- Documentation and Training 7
  - Open Source Declarations 7
  - Training 7
- Getting Help 7
  - Subscribing to Service Notifications 8
- Providing Feedback to Us 8

About This Document **9**
- What's new in this document 9

Overview of vSLX **11**
- Introduction to vSLX 11
- Supported functionality 11
  - SLX 9850 emulation 11
  - SLX-OS features tested 12
  - vSLX limitations 12

vSLX Installation **13**
- Installation overview 13
  - vSLX server requirements 13
- Downloading vSLX software 14
- Installing Linux on an x86 server 14
  - Verifying virtualization support 15
  - Creating additional Linux users 15
- Network-interface setup 15
  - Configuring a static IP address bridge 16
  - Configuring a dynamic IP address bridge 17
- Copying the software distribution 18
- Installing a vSLX lab on a host 19
- Multiple vSLX labs 19
  - Installing the xlxc bash scripts 20
  - Creating a container for vSLX 20
  - Installing vSLX in a container 21
- Upgrading or downgrading vSLX 22
- Uninstalling and reinstalling vSLX 22
- Uninstalling vSLX completely 23

vShell Basics **25**
- Overview of vShell entities 25
  - The vShell environment 25
  - Templates and chassis 25
  - Workstations 25
  - Bridges 25
  - Probes 26
  - Links 26
- Scaling considerations 26

Resource requirements ... 26
Memory swapping ... 26
Virtual machines (VMs) ... 26
SLX 9540 scaling calculations ... 27
SLX 9850 scaling calculations ... 27
Up-and-running example ... 28
Basic vShell tasks ... 30
Starting vShell ... 30
Displaying vShell keyboard shortcuts ... 30
Displaying vShell commands ... 31
Displaying command options ... 31
Creating a template ... 31
Creating a virtual router or switch ... 31
Deleting a virtual router or switch ... 32
Turning on and turning off a virtual device ... 32
Creating a virtual workstation ... 32
Connecting to a virtual device ... 32
Disconnecting from a virtual device ... 33

Virtual Networks ... 35
Overview of virtual networks ... 35
Links ... 35
Creating links between ports ... 35
Deleting links between ports ... 36
Creating and linking a bridge ... 36
Creating and linking a probe ... 37
Linking across servers with tunnels ... 37
Linking to a real interface ... 38

IP Fabric Emulation ... 39
IP Fabric reference topology ... 39
IP Fabric features tested ... 39
IP control-plane features tested ... 39
IP data-plane features tested ... 40

Layer 2 Exchange Emulation ... 41
L2 Exchange reference topology ... 41
Key capabilities ... 41
L2 Exchange features tested ... 42
VPLS features tested ... 42
VLL features tested ... 42
Known L2 Exchange issues ... 42

Advanced vShell Features ... 43
vShell commands from the Linux shell ... 43
User-developed commands and scripts ... 43
Python requirements for extending vShell ... 43
Implementation flow for vShell extensions ... 43
Syntax of the cmdparser( ) function ... 44
Implementing a configuration script ... 45

vShell Command Reference ... 47
clear debug ... 47

connect....49
create bridge....51
create chassis....53
create link....55
create probe....57
create template....59
create tunnel....61
create workstation....63
delete....64
help....66
list....67
poweroff....68
poweron....69
show....70
show debug....73
start....74
system....76
vsh....77
Appendix A: Alternate Routines....79
Multiple vSLX labs (lxc)....79
Creating a container for vSLX (lxc)....79
Installing vSLX in a container (lxc)....82

Preface

- Conventions 7
- Documentation and Training 7
- Getting Help 7
- Providing Feedback to Us 8

Conventions

This section discusses the conventions used in this guide.

Documentation and Training

To find Extreme Networks product guides, visit our documentation pages at:

Current Product Documentation	www.extremenetworks.com/documentation/
Archived Documentation (for earlier versions and legacy products)	www.extremenetworks.com/support/documentation-archives/
Release Notes	www.extremenetworks.com/support/release-notes
Hardware/Software Compatibility Matrices	https://www.extremenetworks.com/support/compatibility-matrices/
White papers, data sheets, case studies, and other product resources	https://www.extremenetworks.com/resources/

Open Source Declarations

Some software files have been licensed under certain open source licenses. More information is available at: www.extremenetworks.com/support/policies/open-source-declaration/.

Training

Extreme Networks offers product training courses, both online and in person, as well as specialized certifications. For more information, visit www.extremenetworks.com/education/.

Getting Help

Support for Extreme vSLX varies from support for other Extreme products.

NOTE

If the source of an issue is common to the latest released version of SLX-OS, then it will be processed according to the customer's SLX-OS product support contract. If the issue is specific to vSLX, Extreme is under no obligation to correct the issue. Extreme periodically evaluates vSLX reported issues, which may be addressed at Extreme's discretion with no specified timeframe.

For support, use one of the following methods:

- Extreme Portal — Search the GTAC knowledge base.
- The Hub — A forum for Extreme Networks customers to connect with one another, answer questions, and share ideas and feedback. This community is monitored by Extreme Networks employees.
- Contact your SE.

If you need to contact your SE, have the following information ready:

- Your Extreme Networks service contract number and/or serial numbers for all involved Extreme Networks products
- A description of the failure
- A description of any action(s) already taken to resolve the problem

Subscribing to Service Notifications

You can subscribe to email notifications for product and software release announcements, Vulnerability Notices, and Service Notifications.

1. Go to www.extremenetworks.com/support/service-notification-form.
2. Complete the form with your information (all fields are required).
3. Select the products for which you would like to receive notifications.

   NOTE
   You can modify your product selections or unsubscribe at any time.

4. Click **Submit**.

Providing Feedback to Us

Quality is our first concern at Extreme Networks, and we have made every effort to ensure the accuracy and completeness of this document. We are always striving to improve our documentation and help you work better, so we want to hear from you! We welcome all feedback but especially want to know about:

- Content errors or confusing or conflicting information.
- Ideas for improvements to our documentation so you can find the information you need faster.
- Broken links or usability issues.

If you would like to provide feedback to the Extreme Networks Information Development team, you can do so in two ways:

- Use our short online feedback form at https://www.extremenetworks.com/documentation-feedback/.
- Email us at documentation@extremenetworks.com.

Please provide the publication title, part number, and as much detail as possible, including the topic heading and page number if applicable, as well as your suggestions for improvement.

About This Document

- What's new in this document..... 9

What's new in this document

This is the first generally available version of the *Extreme vSLX Installation and User Guide*.

> NOTE
> To facilitate discussion of virtual devices running multiple SLX-OS versions, the Release Notes refer to this version of vSLX as vSLX 2.0.0. Otherwise, it's simpler to consider the current vSLX version as 18r.1.00a.

The following table includes descriptions of changes to this guide for the current release.

TABLE 1 Changes in *Extreme vSLX-OS Installation and User Guide, 18r.1.00a*

Feature	Description	Described in
L2 Exchange emulation	Layer 2 Exchange emulation has been verified from this version of vSLX.	Layer 2 Exchange Emulation on page 41

Overview of vSLX

- Introduction to vSLX 11
- Supported functionality 11

Introduction to vSLX

Extreme Virtual SLX (vSLX) is a virtual lab that enables you to emulate ExtremeSwitching SLX 9540 and ExtremeRouting SLX 9850 devices. You can also create virtual networks of workstations, SLX devices, tunnels, bridges, and probes.

You can use vSLX for training, configuration buildout and validation, workflow and automation development, and testing. For example:

- Hands-on training of SLX-OS for SLX 9540 and SLX 9850 CLI and programmatic API
- Building and validation of configuration before applying it to a supported device
- Development and testing of automation scripts and software, independent of hardware
- Development and testing Extreme Workflow Composer (EWC) workflows
- Configuration of management plane functionality (CLI and programmatic API) for SLX-OS features

There are two supported installation contexts:

- Host installation: All users share one virtual lab.
- Installation in Linux containers: Multiple users have independent virtual labs.

Supported functionality

Under vSLX, you can emulate most SLX 9540 and SLX 9850 functionality.

This document supports emulating SLX 9540 and SLX 9850 devices running the Extreme SLX-OS version indicated on the cover page. If you need to emulate these devices running a different SLX-OS version, refer to the Release Notes.

> NOTE
> After installation and basic configuration, the **show templates** command displays the installed version of SLX-OS. For details, refer to Up-and-running example on page 28 or show on page 70.

Control-plane and configuration-processing emulations are supported.

Although L2/L3 and single-VTEP VXLAN data planes are supported, data-plane emulation does not support heavy traffic for data-plane service deployment. The data plane is supported only for verification of control-plane deployment using ping, traceroute, and so forth.

All basic SLX 9540 hardware elements can be emulated.

SLX 9850 emulation

A virtual SLX 9850 device consists of one management module (MM) and multiple line cards (LCs). There is one management module virtual machine (MMVM) and a line card virtual machine (LCVM) for each line card. All VMs belonging to a chassis are connected by a virtual backplane, over which they communicate.

The following SLX 9850-4 elements can be emulated:

- One management module (MM)
- Control plane
- Any combination of as many as four of the following line cards:
  - 72-port 10 GbE
  - 36-port 100 GbE

The following SLX 9850-8 elements can be emulated:

- One management module (MM)
- Control plane
- Any combination of as many as eight of the following line cards:
  - 72-port 10 GbE
  - 36-port 100 GbE

SLX-OS features tested

The following infrastructure and control-plane features have been tested for this release:

- Basic infrastructure
- BFD
- BGP and BGP IPv6
- ISIS and ISIS IPv6
- L3VPN
- LAG (port-channels)
- LDP
- MPLS-TE
- OSPF and OSPFV3
- Service and Monitoring (syslog, SNMPWALK, and SNMP traps)
- STP, RSTP, and MSTP
- VLAN
- VRRP and VRRPv3

For IP Fabric features tested for this release, refer to IP Fabric features tested on page 39.

vSLX limitations

Cloud deployment of vSLX has not been tested.

vSLX supports network packet broker (NPB) system-mode only for exercising the command and control portion of the network. Therefore, solution support is limited to Extreme Visibility manager and leveraging or verification of SSH, SNMP, Syslog, and OpenFlow. Integration of vSLX and SessionDirector is not currently emulated.

vSLX Installation

- Installation overview 13
- Downloading vSLX software 14
- Installing Linux on an x86 server 14
- Network-interface setup 15
- Copying the software distribution 18
- Installing a vSLX lab on a host 19
- Multiple vSLX labs 19
- Upgrading or downgrading vSLX 22
- Uninstalling and reinstalling vSLX 22
- Uninstalling vSLX completely 23

Installation overview

Installation of vSLX requires the following top-level tasks.

NOTE
Although running vSLX does not require Linux experience, installing vSLX does require basic Linux experience.

TABLE 2 Installation tasks

Task	Topics
Download software	Downloading vSLX software on page 14
Install Ubuntu Linux	Installing Linux on an x86 server on page 14
Create an IP-address bridge	Perform one of the following tasks: • Configuring a static IP address bridge on page 16 • Configuring a dynamic IP address bridge on page 17
Copy downloaded software	Copying the software distribution on page 18
Install vSLX	Perform one of the following tasks: • Installing a vSLX lab on a host on page 19 • Multiple vSLX labs on page 19

vSLX server requirements

vSLX is hosted on an x86 server platform; server requirements depend on the scale of the deployment.

At the low end, a laptop with 12 GB of memory can be used to emulate a single SLX 9540 device. For emulation of an SLX 9850 device or multiple-device networks, install vSLX on a server. The following table lists the CPU, memory, and storage requirements of a vSLX server.

TABLE 3 vSLX server requirements

Scope	CPUs	Memory required	Storage required
One device	i5 (4 cores)	12 GB	40 GB (SLX 9540) 150 GB (SLX 9850-4)

TABLE 3 vSLX server requirements (continued)

Scope	CPUs	Memory required	Storage required
Network	E5 (or better) Minimum: 12 cores Recommended: 24–32 or more cores	Minimum: 64 GB Recommended: 128 GB	1 TB

For scoping details, refer to Scaling considerations on page 26.

Downloading vSLX software

Download the required software components to a folder named `vslx-rel` on your workstation.

NOTE
For download-URL details, please refer to the Release Notes for this product.

TABLE 4 Files required to download from Extreme

File	Notes
vslx-x.x.x.tar.gz	Contains the vSLX distribution.
slxos-version.x.xx.tar.gz	Contains the Extreme SLX-OS for SLX 9540 and SLX 9850 distribution.

Installing Linux on an x86 server

vSLX runs on Ubuntu Linux, installed on an x86 server.

Note the following installation guidelines:

- The appropriate Ubuntu version is Ubuntu Server 16.04.x LTS, which you can download from https://www.ubuntu.com/.
- You can either install Ubuntu as a clean installation or upgrade from a previous version.
- Include the OpenSSH server package in your installation.
- For detailed instructions, refer to the Ubuntu website.

The top-level steps are as follows.

1. Create a bootable USB stick by using the Ubuntu ISO image. Click the following link for detailed instructions: https://www.ubuntu.com/download/desktop/create-a-usb-stick-on-ubuntu.
2. Boot your server from the USB stick and follow the instructions to install the Ubuntu distribution on the hard disk.
3. Log in with the user and password you created during installation, and verify that you have `sudo` privileges.

Verifying virtualization support

You need to verify that your x86 computer is configured to support hardware virtualization.

NOTE
The syntax used in this task is valid for Intel x86 CPUs. For other supported CPUs, refer to the appropriate resources.

1. Log in to Linux, and enter the following command.

```
user@ubuntu: cat /proc/cpuinfo | grep vmx
```

   If the output contains `vmx` flags, your system is already configured for hardware virtualization.

2. If the output does not contain `vmx` flags:
   a) Reboot the computer and open the BIOS menu.
   b) Enable hardware virtualization, save, and exit.
   c) Log in to Linux, and enter the following command to verify that virtualization is now enabled.

```
user@ubuntu: cat /proc/cpuinfo | grep vmx
```

   If the output contains `vmx` flags, your system is now configured for hardware virtualization.

Creating additional Linux users

If needed, create additional users with Linux sudo privileges on the x86 server.

If you will be implementing Multiple vSLX labs on page 19, best practice is to create a user for each container-based virtual lab.

1. Create a user.

```
user@ubuntu:~$ sudo adduser --shell /bin/bash --home /home/vlab1_user vlab1_user
```

2. When prompted, set the password.

```
Enter new UNIX password:
**********
```

3. Add the user to the `sudo` group.

```
user@ubuntu:~$ sudo usermod -a -G sudo vlab1_user
```

4. To verify that the new user is created and belongs to the `sudo` group, enter the **sudo groups** command.

```
user@ubuntu:~$ sudo groups vlab1_user
```

Network-interface setup

You need to create a bridge to connect the server external management interface and the vSLX management interface.

The bridge implementation varies if Dynamic Host Configuration Protocol (DHCP) is employed or not:

- If you are not using DHCP, perform Configuring a static IP address bridge on page 16.
- If you are using DHCP, perform Configuring a dynamic IP address bridge on page 17.

Configuring a static IP address bridge

If your network configuration is not controlled by a DHCP server, perform this task to configure a static IP address bridge.

1. (If required) To retrieve the IP address and the netmask, enter **ifconfig** *interface_name*.
2. Enter **ifconfig -a** to verify that the Ethernet interface is up.

```
user@ubuntu:~$ ifconfig -a
```

3. Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
user@ubuntu:~$ ping example.com
```

4. Install `bridge-utils`, using the following commands:

```
user@ubuntu:~$ sudo apt-get update
user@ubuntu:~$ sudo apt-get install bridge-utils
```

5. Enter **sudo vi /etc/network/interface** to modify the following `interfaces` example with the values you retrieved, replace "eth0" with the actual server external interface, and replace all other < > with your values.

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 0.0.0.0

auto br0
iface br0 inet static
address <ip-address>
netmask <net-mask>
gateway <gateway-ip-address>
dns-nameserver <dns-nameservers>
dns-search <search-domains>
bridge_ports <eth0>
bridge_stp off
bridge_fd 0
bridge_maxwait 0
```

6. Save and close the file.
7. Reboot the server.

```
user@ubuntu:~$ sudo reboot
```

Configuring a dynamic IP address bridge

If your network configuration is controlled by a DHCP server, perform this task to configure a dynamic IP address bridge.

1. To display the available network interfaces, enter **cat /etc/network/interfaces**.

```
user @ubuntu:~$ cat /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens3
iface ens3 inet dhcp
```

2. Enter **ifconfig -a** to verify that the Ethernet interface is up.

```
user@ubuntu:~$ ifconfig -a
```

3. Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
user@ubuntu:~$ ping example.com
```

4. Install `bridge-utils`, using the following commands:

```
user@ubuntu:~$ sudo apt-get update
user@ubuntu:~$ sudo apt-get install bridge-utils
```

5. Enter **sudo vi /etc/network/interface** to modify the `interfaces` file.

```
user@ubuntu:~$ sudo vi /etc/network/interface
```

6. Define the `br0` bridge and enslave the primary network interface (`ens` in the following example) under `br0`.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
# We commented out the following two lines:
#auto ens3
#iface ens3 inet dhcp

# We added the following definition of br0:
auto br0
iface br0 inet dhcp
        bridge_ports ens3
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

7. Save and close the file.

8. Reboot the server.

```
user@ubuntu:~$ sudo reboot
```

9. After logging back in, enter **ifconfig -a** to verify that the IP address is now assigned to br0.

```
user@ubuntu:~$ ifconfig -a
br0       Link encap:Ethernet  HWaddr 52:55:00:d1:55:01
          inet addr:10.0.0.110  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2601:644:8780:2e56:e99d:2411:3c93:b7e6/64 Scope:Global
          inet6 addr: 2601:644:8780:2e56:5055:ff:fed1:5501/64 Scope:Global
          inet6 addr: fe80::5055:ff:fed1:5501/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:142 errors:0 dropped:0 overruns:0 frame:0
          TX packets:93 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:18815 (18.8 KB)  TX bytes:13887 (13.8 KB)

ens3      Link encap:Ethernet  HWaddr 52:55:00:d1:55:01
          inet6 addr: fe80::5055:ff:fed1:5501/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:142 errors:0 dropped:0 overruns:0 frame:0
          TX packets:103 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20803 (20.8 KB)  TX bytes:14793 (14.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:80 errors:0 dropped:0 overruns:0 frame:0
          TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:5920 (5.9 KB)  TX bytes:5920 (5.9 KB)
```

Copying the software distribution

Copy the downloaded files from your workstation to the Linux server on which you are installing vSLX, and list them.

1. On the Linux server, create an `slxos-dist` directory and change to it.

```
user@ubuntu:~$ sudo mkdir /slxos-dist
user@ubuntu:~$ cd /slxos-dist
```

2. Use the **scp** command to copy the compressed distribution files from your workstation to the `slxos-dist` directory on the Linux server; then list them.

> NOTE
> The file names in this flow are for release 17r.2.00. Substitute the names of the actual files you downloaded in Downloading vSLX software on page 14.

```
user@ubuntu:/slxos-dist$ sudo scp user@192.0.2.0:/home/user/work/vslx-rel/vslx2.0.0.tar.gz .
user@ubuntu:/slxos-dist$ sudo scp user@192.0.2.0:/home/user/work/vslx-rel/slxos17r.2.00.tar.gz .
user@ubuntu:/slxos-dist$ ls
slxos17r.2.00.tar.gz  vslx2.0.0.tar.gz
```

3. Unpack the SLX-OS distribution and then display the contents of the `slxos-dist` directory.

```
user@ubuntu:/slxos-dist$ sudo tar -zxf slxos17r.2.00.tar.gz
user@ubuntu:/slxos-dist$ ls
slxos17r.2.00  slxos17r.2.00.tar.gz  vslx2.0.0.tar.gz
```

4. Display the contents of the release directory (for example, `slxos17r.2.00`).

```
user@ubuntu:/slxos-dist$ ls slxos17r.2.00
app_names                        onie-host-installer      SWBD2001
breeze_tools                     onie-installer           SWBD2002
breeze_tools.tgz                 onie-installer-files.tgz SWBD2003
Brocade_SLX_CatenatedMIBs.tar.gz onie-target              SWBD2016
brocade_slx_proto_models.tar.gz  oss-binaries-x86_64-ubuntu-14.04-rootfs.tar.gz  SWBD2017
common                           platform_names           SWBD2500
install                          pubkey.pem               SWBD2900
install_cr                       signature2.tar           SWBD4000
install_pbr                      signature2.tar.sig       vnos-deinstall
install_verify                   signature.tar            vnos-install
libbrcmsdk.a.SWBD2000.tgz        signature.tar.sig        vslx-utils_1.0.0.deb
libbrcmsdk.a.SWBD2016.tgz        slx17r200_yang.tar.gz
libbrcmsdk.a.SWBD4000.tgz        SWBD2000
```

5. Unpack the vSLX distribution and then display the contents of the `slxos-dist` directory.

```
user@ubuntu:/slxos-dist$ sudo tar -zxf vslx2.0.0.tar.gz
user@ubuntu:/slxos-dist$ ls
extreme-lxc_2.0.0.deb  slxos17r.2.00  slxos17r.2.00.tar.gz  vslx2.0.0.tar.gz  vslx_2.0.0.deb
```

Installing a vSLX lab on a host

Perform this task to install a shared virtual lab on an x86 server.

> NOTE
> If you require independent virtual labs on a server, refer to Multiple vSLX labs on page 19.

The vSLX host distribution package contains the software components required to virtualize supported SLX devices on a x86 server.

1. Enter the **dpkg** command to install the vSLX Debian package on the host.

```
user@ubuntu:~$ sudo dpkg -i /slxos-dist/vslx_2.0.0.deb
```

2. Run the **postinst-setup.sh** utility to install the dependent packages and configure system files.

```
user@ubuntu:~$ /VM/postinst-setup.sh
```

After successful installation, proceed to Up-and-running example on page 28.

Multiple vSLX labs

Installing vSLX in Linux containers—rather than directly on a host—enables each user to have an independent virtual lab.

Linux Containers (LXC) support running multiple Linux systems on a control host using a single Linux kernel. We use *privileged* containers, created by root and running as root. After creating containers, you install one vSLX virtual lab in each container.

The recommended implementation flow utilizes a Linux bash script that we have developed. (This script enables **xlxc** commands, based on the Linux **lxc** commands.):

- (Once) Installing the xlxc bash scripts on page 20
- (For each lab) Creating a container for vSLX on page 20
- (For each lab) Installing vSLX in a container on page 21

The advantages of the **xlxc** flow over directly using **lxc** are as follows:

- There is no need to edit the default Linux container-configuration file.
- The command syntax is less complex.
- There are fewer steps.

NOTE
If you need container installation of vSLX using **lxc** commands (and manual editing of the container-configuration file), refer to Multiple vSLX labs (lxc) on page 79.

Installing the xlxc bash scripts

This task is a prerequisite for the recommended flow of installing and managing vSLX containers.

NOTE
You need to perform this task only once.

1. Log in to the x86 server as a user with sudo privileges.

   NOTE
   Although all sudo users can create and access all containers, our user in this flow is `vlab1_user`, created in Creating additional Linux users on page 15.

2. Install **lxc** using the following commands, confirming prompts to continue.

```
vlab1_user@ubuntu:~$ sudo apt-get update
vlab1_user@ubuntu:~$ sudo apt-get install lxc
```

3. Install the **xlxc** bash scripts, which are wrappers over the Linux **lxc** commands.

```
vlab1_user@ubuntu:~$ sudo dpkg -i /slxos-dist/extreme-lxc_2.0.0.deb
```

4. For help on the **xlxc** options:
   - For a list of sub-commands, enter **xlxc help**.

```
vlab1_user@ubuntu:~$ xlxc help
```

   - For help on a sub-command, enter **xlxc help <sub-command>**.

```
vlab1_user@ubuntu:~$ xlxc help create
```

Creating a container for vSLX

(For the **xlxc** flow only) This task is a prerequisite for installing each container instance of vSLX.

1. Make sure that you are logged in to the x86 server as a user with sudo privileges.

2. Enter the **xlxc create** command to create the container.

```
vlab1_user@ubuntu:~$ xlxc create VLAB1
  Using image from local cache
  Unpacking the rootfs
---
  You just created an Ubuntu xenial amd64 (20180701_07:42) container.
  To enable SSH, run: apt install openssh-server
  No default root or user password are set by LXC.
  Container config is patched
  Container root account is enabled with no password
  Container VLAB1 is created successfully
```

3. Enter the **xlxc mount** command.

```
vlab1_user@ubuntu:~$ xlxc mount VLAB1 /slxos-dist /slxos-dist
Host '/slxos-dist' is mounted on '/slxos-dist' in container 'VLAB1'
```

Installing vSLX in a container

(For the **xlxc** flow only) After preparing a Linux container, perform this task to installa vSLX virtual lab in the container.

1. Start the container and log in as root (with no password required).

```
vlab1_user@ubuntu:~$ xlxc start VLAB1
  Connected to tty 0
  Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself
(output
              truncated)
  Ubuntu 16.04.4 LTS VLAB1 console
VLAB1 login:
```

2. Verify that all is ready for vSLX installation, as follows:
   a) List the `slxos-dist` directory.

```
root@VLAB1:~# ls /slxos-dist
extreme-lxc_2.0.0.deb  slxos17r.2.00  slxos17r.2.00.tar.gz  vslx_2.0.0.deb  vslx2.0.0.tar.gz
```

   NOTE
   You can also list included directories.

   b) Verify that the container's Ethernet interface is up and that an internet address is assigned.

```
root@VLAB1:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:16:3e:b5:7d:b9
          inet addr:192.0.2.0  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2001:db8::/64 Scope:Link
          inet6 addr: 2001:db8:ffff:ffff:ffff:ffff:ffff:ffff/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:59 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9877 (9.8 KB)  TX bytes:1312 (1.3 KB)
(output
               truncated)
```

   c) Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
root@VLAB1:~# ping example.com
```

3. Install the vSLX debian package.

```
root@VLAB1:/slxos-dist# sudo dpkg -i /slxos-dist/vslx_2.0.0.deb
Selecting previously unselected package vslx.
(Reading database ... 13616 files and directories currently installed.)
Preparing to unpack ./vslx_2.0.0.deb ...
Unpacking vslx (2.0.0) ...
Setting up vslx  (2.0.0) ...
```

4. Run the post-installation script.

```
root@VLAB1:~# /VM/postinst-setup.sh
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Fetched 204 kB in 1s (202 kB/s)
Reading package lists... Done
Reading package lists... Done
```

After successful installation, proceed to Up-and-running example on page 28.

Upgrading or downgrading vSLX

For instructions on how to upgrade or downgrade vSLX, please refer to the Release Notes.

Uninstalling and reinstalling vSLX

Use this task to uninstall a vSLX instance—preserving your vSLX objects and configurations—with an option to reinstall vSLX.

NOTE

The prompts below follow Installing a vSLX lab on a host on page 19. The flow is similar following Multiple vSLX labs on page 19.

1. Although this task automatically closes any vSLX VMs, close any other VMs.
2. Uninstall the vSLX distribution—preserving your vSLX objects and configurations.

```
user@ubuntu:~$ sudo dpkg -r vslx
```

3. NOTE

   To install a new version of vSLX—preserving vSLX objects and configurations from a previous installation—refer to the Release Notes for the new version.

   To reinstall vSLX:

   a) Enter the dpkg -i command, specifying the relevant vSLX package.

```
user@ubuntu:~$ sudo  dpkg -i vslx_x.x.x.deb
```

   b) Enter the **postinst-setup.sh** command.

```
user@ubuntu:~$ sudo /VM/postinst-setup.sh
```

Uninstalling vSLX completely

Use this task to completely uninstall a vSLX instance, including your vSLX objects and configurations.

NOTE
The prompts below follow Installing a vSLX lab on a host on page 19. The flow is similar following Multiple vSLX labs on page 19.

1. Although this task automatically closes any vSLX VMs, close any other VMs.
2. Log in to vShell.

```
user@ubuntu:~$ vsh
```

3. Power off all vSLX devices.

```
(vsh) poweroff ch_9850_4_02
(vsh) poweroff ch_9540_01
```

4. Delete all vSLX devices.

```
(vsh) delete chassis ch_9850_4_02
(vsh) delete chassis ch_9540_01
```

5. Delete all vSLX templates.

```
(vsh) delete template 9850_templ
(vsh) delete template 9540_templ
```

6. Exit vShell.

```
(vsh) exit
```

7. Uninstall the vSLX distribution.

```
user@ubuntu:~$ sudo /VM/uninstall.sh
```

8. If needed, delete users that you created.

```
user@ubuntu:~$ sudo deluser testuser
```

vShell Basics

- Overview of vShell entities....25
- Scaling considerations....26
- Up-and-running example....28
- Basic vShell tasks....30

Overview of vShell entities

Extreme Virtual Shell (vShell) commands, objects, and relationships enable you to create and manage supported virtual devices and networks.

The vShell environment

Virtual Shell (vShell) is a Python-based interactive environment that enables you to create and manage virtual system instances.

NOTE
For documentation of vShell commands, refer to vShell Command Reference on page 47.

Templates and chassis

A *template* object is a virtual image of an Extreme SLX-OS device. A *chassis* is a bootable instance of a template that represents a device.

The current vSLX version supports two template types:

- SLX 9540
- SLX 9850

Chassis-creation flow is as follows:

1. Create named templates from one or both of the supported template types.
2. Create one or more chassis (devices), each specifying a named template.

When you create chassis from SLX 9850 templates, you specify line-card types and numbers.

Workstations

You can also create *workstations* (virtual PCs). In general, you virtually connect workstations to virtual devices (chassis) that you created.

The current version supports only workstations running Ubuntu Linux.

Bridges

You can connect a port to no more than one other port. A bridge enables you to connect multiple ports, forwarding traffic received by one port to all ports connected to the bridge.

NOTE
A virtual bridge does not learn MAC addresses.

Probes

To connect a probe to an entity—a device port, a bridge, or a host interface—use the **link -probe** command. Once the connection is created, traffic to and from the entity is forwarded to the probe. You can use x86 network tools to monitor probed traffic.

Links

A link is a virtual cable connecting two ports, a port and an entity, or two entities.

Scaling considerations

This section contains information that helps you estimate CPU, RAM, and disk requirements for vSLX entities.

This information is valid for both installation options:

- Installing a vSLX lab on a host on page 19
- Multiple vSLX labs on page 19

NOTE
For server CPU, memory, and storage specifications, refer to vSLX server requirements on page 13.

Resource requirements

The following table lists virtual CPU (vCPU), memory, and storage requirements needed to emulate SLX-OS entities.

TABLE 5 Resources required for vSLX entities

Entity	CPU cores	Virtual memory (RAM)	Min. disk size	Max. disk size
SLX 9540 template	—	—	9 GB	9 GB
SLX 9540 instance	1	8 GB	1 GB	25 GB
SLX 9850 template	—	—	20 GB	20 GB
SLX 9850 MM	1	8 GB	1 GB	25 GB
SLX 9850 line card	1	4 GB	1 GB	25 GB

Memory swapping

If your storage technology is solid-state drive (SSD)—rather than rotational hard-disk drive (HDD)—memory swapping into the SSD can reduce the physical "Virtual memory (RAM)" values in the preceding table. For example, a swap space of 32 GB supports four additional SLX 9540 instances.

Virtual machines (VMs)

Each of the following entities is emulated by one virtual machine (VM):

- SLX 9540 instance

- SLX 9850 management module (MM)
- SLX 9850 line card (LC)

vSLX uses several methodologies to optimize resource utilization among VMs:

- Space efficiency: QEMU snapshot technology is used for chassis (virtual device) image creation from a template. When a chassis is created, the initial storage size is less than 200 KB. As a chassis is configured, the disk size increases.
- Memory efficiency: QEMU and kernel same-page merging (KSM) support memory de-duplication among VMs, which enables sharing memory for duplicate blocks. These technologies reduce memory requirements by up to 60 percent.
- CPU over-provisioning: When vSLX is used for normal control-plane traffic, CPU consumption does not exceed two vCPUs for each VM. This efficiency allows for flexibility in over-subscribing CPU resources, allowing vSLX to scale well.

SLX 9540 scaling calculations

The following equations estimate optimized server resources needed for virtual SLX 9540 devices (each SLX 9540 instance is one VM):

- $CPU_{SLX\ 9540}$ (number of cores) = 1 * <number-of-VMs>
- $MEM_{SLX\ 9540}$ (GB) = 8 * <number-of-VMs>
- $STORAGE_{SLX\ 9540}$ (GB) = $TemplateSize_{SLX\ 9540}$ + (2 * <number-of-VMs>)

The following examples estimate the server resources needed to emulate 10 SLX 9540 instances:

- CPU = 1 * 10 = 10 cores
- MEM = 8 * 10 = 80 GB
- STORAGE = 6 + (2 x 10) = 26 GB

SLX 9850 scaling calculations

The following equations show how to estimate optimized server resources needed for virtual SLX 9850 devices.

> NOTE
> Each SLX 9850 instance has one management-module VM (MMVM) and one line-card VM (LCVM) for each line card.

- $CPU_{SLX\ 9850}$ (number of cores) = 1 * (<number-of-MMVMs> + <number-of-LCVMs>)
- $MEM_{SLX\ 9850}$ (GB) = (8 * <number-of-MMVMs>) + (4 * <number-of-LCVMs>)
- $STORAGE_{SLX\ 9850}$ (GB) = $TemplateSize_{SLX\ 9850}$ + 2 * (<number-of-MMVMs> + <number-of-LCVMs>)

The following examples show how to estimate the server resources needed to emulate three SLX 9850-4 instances. (Each instance has three line cards installed, for a total of nine line cards.)

- CPU = 1 * (3 + 9) = 1 * 12 = 12 cores
- MEM = (8 * 3) + (4 * 9) = 60 GB
- STORAGE = 16 + (2 * 12) = 16 + 24 = 40 GB

Up-and-running example

This example logs you in to vShell, creates a template and two linked devices; turns on, connects to, logs in to the devices, and defines IP addresses; and logs out and disconnects from the devices.

1. From Linux, log in to vShell.
   - On a host-based vSLX installation:

     ```
     user@ubuntu:~$ vsh
      For help anytime, type '?'
      When you're done, type '^D'
     (vsh)
     ```

   - On a container-based vSLX installation:

     ```
     root@VLAB1:~# vsh
      Bridge address  : 10.0.0.64/24
      Default gateway : 10.0.0.1
      For help anytime, type '?'
      When you're done, type '^D'
     (vsh)
     ```

2. Enter the **create template** command.

   ```
   (vsh) create template TMPL1-AV slx9540 /slxos-dist
   Building SLX9540 Image ...

   ***** Created VM PB4000 *****
   PB4000 created.
   ```

3. To verify template creation, enter the **show templates** command.

   ```
   (vsh) show templates
   Name        Type     SLXOS Version     Snapshots
   ..............................................
   TMPL1-AV    slx9540  slx-os17r.2.00    0
   ```

4. Create two virtual SLX 9540 devices, using the **create chassis** command.

   ```
   (vsh) create chassis av1 1U TMPL1-AV
   Domain PB_1_av1 defined from /VM/templates/snapsh/av1/BVM_PB_SIM_1.xml
   (vsh) create chassis av2 1U TMPL1-AV
   Domain PB_1_av2 defined from /VM/templates/snapsh/av2/BVM_PB_SIM_1.xml
   ```

5. Create a Linux workstation, using the **create workstation** command.

   ```
   (vsh) create workstation h1 linux
   Converting qcow2 image to raw format.
   Please wait a few seconds...Done
   /VM/templates/host/snapsh/h1/HOST_VM_DEF_S.xml
   Domain h1 defined from /VM/templates/host/snapsh/h1/HOST_VM_DEF_S.xml
   ```

6. Enter the **show system** command.

   ```
   (vsh) show system
    Device Name             Type            Power
    ..................................................
    av1                     SLX 9540        off
    av2                     SLX 9540        off
    h1                      VPC             off
   ```

7. Enter the **poweron** command to turn on the devices.

```
(vsh) poweron av1
 The device is powered on successfully (1)
(vsh) 2 poweron av2
 The device is powered on successfully (1)
(vsh) poweron h1
```

8. Enter the **show system** command.

```
Device Name             Type            Power
.................................................
av1                     SLX 9540        on
av2                     SLX 9540        on
h1                      VPC             on
```

9. Create a link between the devices.

```
(vsh) create link av1 0/1 av2 0/1
```

NOTE

If you were to run **show link** and **show link status** at this point, they would display "Down" and "Both sides down", respectively. The link will be up only after you log in to SLX-OS and configure av1 0/1 and av2 0/1, as described below.

10. Connect to one of the devices, log in to SLX-OS, and assign an IP address.

```
(vsh) connect av1
SLX-OS (SLX)SLX login: admin
password: password
device# configure terminal
device(config)# interface ethernet 0/1
device(conf-if-eth-0/1)# ip address 10.0.0.1/24
device(conf-if-eth-0/1)# no shut
```

11. Log out from the first device and return to vShell.

```
device(conf-if-eth-0/1)# end
device# exit
SLX-OS (SLX) Ctrl-]
(vsh)
```

12. Connect to the second device, log in to SLX-OS, and assign an IP address.

```
(vsh) connect av2
SLX-OS (SLX)SLX login: admin
password: password
device# configure terminal
device(config)# interface ethernet 0/1
device(conf-if-eth-0/1)# ip address 10.0.0.2/24
device(conf-if-eth-0/1)# no shut
```

13. Log out from the second device and return to vShell.

```
device(conf-if-eth-0/1)# end
device# exit
SLX-OS (SLX) Ctrl-]
(vsh)
```

14. To verify the link between the two devices, enter the **show link** command.

```
(vsh) show link
 Name           Port        Name           Port           State
 ..................................................................
 av1             0/1    <--> av2             0/1            Up
```

15. To turn off the devices, enter the **poweroff** command.

```
(vsh) poweroff av1
Stopping gos servers for chassis av1
 The device is powered off successfully (1)
(vsh) poweroff av2
Stopping gos servers for chassis av2
 The device is powered off successfully (1)
(vsh) poweroff h1
```

Basic vShell tasks

Perform these tasks to log in to vShell, access CLI Help, and configure virtual devices.

NOTE

For additional details of vShell commands, refer to vShell Command Reference on page 47.

Starting vShell

NOTE

If needed, you can open multiple vShell sessions from different Linux shells.

To start vShell, enter **vsh**.

```
$ vsh
 For help anytime, type '?'
 When you're done, type '^D'
(vsh)
```

Displaying vShell keyboard shortcuts

To display vShell keyboard shortcuts, enter **help**.

```
(vsh) help
  ?  - for help
 Tab - auto fill
 ^P  - previous command
 ^N  - next command
 ^D  - end of the session
 ^A  - move cursor to the beginning
 ^E  - move cursor to the end
 ^W  - delete a word
 ^U  - delete from the beginning
 ^K  - delete to the end
```

Displaying vShell commands

To display available commands, enter a question mark (**?**).

```
(vsh) ?
 clear                  Clear
 connect                Connect console
 create                 Provisioning
 delete                 Destroy
 exit                   Exit shell
 help                   Get help
 list                   Show running devices
 poweroff               Power off device
 poweron                Power on device
 show                   Show info
 start                  Power on and connect
 system                 Bridge daemon control
```

Displaying command options

To display command parameters, type the command, press the Space key, and then type a question mark (**?**) .

```
(vsh) create chassis link ?
 -bridge                Bridge
 -interface             Interface
 -probe                 Probe
 String                 Device name
```

Creating a template

To create a device template, enter the **create template** command.

> NOTE
> For details, refer to create template on page 59.

The following example shows how to create a template of the SLX 9850 type.

```
(vsh) create template 9850_templ slx9850 /slxos-dist
```

Creating a virtual router or switch

To create a virtual router or switch, enter the **create chassis** command.

> NOTE
> For details, refer to create chassis on page 53.

The following example shows how to create a virtual SLX 9850-4 router, with a management module (MM) and three line cards. Slots 1 and 4 have the 72x10G line-card type and slot 2 has a 36x100G line card. This installation takes approximately 15 minutes.

```
(vsh) create chassis ch_9850_4_01 f4 1 1/72x10G, 2/36x100G,4/72x10G 9850_templ
```

Deleting a virtual router or switch

To delete a virtual router or switch, enter the **delete chassis** command. The following example includes two devices, test1 and test2. After delete test1, the **show chassis** command displays only test2.

```
(vsh) show chassis
 Chassis Name            Type            Power
 ...................................................
 test1                   SLX 9850        on
 test2                   SLX 9850        off

(vsh) delete chassis test1

(vsh) show chassis
 Chassis Name            Type            Power
 ...................................................
 test2                   SLX 9850        off
```

Turning on and turning off a virtual device

To turn on a virtual router, switch, or workstation, enter the **poweron** command. To turn off a virtual router, switch, or workstation, enter the **poweroff** command .

```
(vsh) show system
 Device Name             Type            Power
 ...................................................
 av1                     SLX 9540        off
 av2                     SLX 9540        off
 F1                      SLX 9850        off
 F2                      SLX 9850        off
 F3                      SLX 9850        off
 host1                   VPC             off
 host2                   VPC             off

(vsh) poweron F3

(vsh) show system
 Device Name             Type            Power
 ...................................................
 av1                     SLX 9540        off
 av2                     SLX 9540        off
 F1                      SLX 9850        off
 F2                      SLX 9850        off
 F3                      SLX 9850        on
 host1                   VPC             off
 host2                   VPC             off
```

Creating a virtual workstation

To create a virtual PC workstation, enter the **create workstation** command.

```
(vsh) create workstation linux_ws_01 linux
```

Connecting to a virtual device

To connect the current `vsh` session to the console of a virtual workstation, chassis, management module, or line card, enter the **connect** command.

```
(vsh) connect test1

SLX-OS (SLX)
SLX login:
```

Disconnecting from a virtual device

To disconnect from a vSLX virtual device, press **Ctrl-]**. (Press the **]** key while holding down the **Ctrl** key.)

Virtual Networks

- Overview of virtual networks....35
- Links....35
- Creating and linking a bridge....36
- Creating and linking a probe....37
- Linking across servers with tunnels....37
- Linking to a real interface....38

Overview of virtual networks

Under vSLX, you can create and connect virtual SLX devices that simulate a network.

You can also create and connect virtual workstations, bridges, probes, and tunnels, setting up a virtual network lab.

NOTE
For information regarding VLABs with objects from multiple SLX-OS releases, please refer to the Release Notes.

Links

A link is a virtual network cable that connects two device ports or other network entities.

The topics in this section explain how to create and delete links between two device ports. You can also link the following entities:

- Bridges
- Probes
- Workstations
- Tunnels

For configuration details of all these entities, refer to create link on page 55.

Creating links between ports

Use this task to create a virtual Ethernet cable between two ports.

The steps in this task create two links between a SLX 9540 device and a SLX 9850-8 device.

1. Create the first link.

```
(vsh) create link ch_9540_01 0/1 ch_9850_8_01 1/1
```

2. Create the second link.

```
(vsh) create link ch_9540_01 0/3 ch_9850_8_01 5/1
```

3. (Optional) To verify link status, run the **show link status** command.

```
(vsh) show link status
 Name           Port        Name           Port           Status
 ..............................................................................
 ch_9540_01     0/13   <--> ch_9850_8_01   1/1            Right side down
 ch_9540_01     0/12   <--> ch_9850_8_01   5/1            All up
```

Deleting links between ports

Use this task to delete virtual links between ports.

The steps in this task delete two links between a SLX 9540 device and a SLX 9850-8 device.

1. Delete the first link.

```
(vsh) delete link ch_9540_01 0/1 ch_9850_8_01 1/1
```

2. Delete the second link.

```
(vsh) delete link ch_9540_01 0/3 ch_9850_8_01 5/1
```

Creating and linking a bridge

Use this task to create a bridge and link it to devices.

A bridge enables you to connect a port to more than one additional port or other entity. The bridge forwards all traffic received by one entity to all other entities connected to the bridge.

1. Enter the **create bridge** command.

```
(vsh) create bridge BR1
```

2. Enter the required **create link** commands.

```
(vsh) create link -bridge BR1 RT1 4/10
(vsh) create link -bridge BR1 RT2 5/12
(vsh) create link -bridge BR1 host1 0/1
```

The following example shows how to create a bridge, link two routers and a host through the bridge, display the links, and display the bridge summary.

```
(vsh) create bridge BR1
(vsh) create link -bridge BR1 RT1 4/10
(vsh) create link -bridge BR1 RT2 5/12
(vsh) create link -bridge BR1 host1 0/1
(vsh) show link
 Name           Port        Name           Port           State
 ..............................................................................
 RT1            4/10   <--> BR1          [bridge]       Up
 RT2            5/12   <--> BR1          [bridge]       Up
 host1          0/1    <--> BR1          [bridge]       Up

(vsh) show bridge
 Name           No.connections
 ..................................
 BR1            3
```

Creating and linking a probe

Use this task to create a probe and link it to devices.

When you link a probe to a network entity, traffic sent from or received by the entity is forwarded to the probe. You can then use host utitilies like tcpdump to monitor the traffic.

1. Enter the **create probe** command.

```
(vsh) create probe PB2
```

2. Enter the required **create link** command.

```
(vsh) create link -probe PB1 RT3 1/4
```

The following example shows how to create a probe, link it to a device port, display the links, and display the probe summary.

```
(vsh) create probe PB2
vsh) create link -probe PB1 RT3 1/4
(vsh) show link
 Name           Port        Name           Port           State
 ..............................................................
 RT1            1/2    <--> RT3            1/4            Up
 RT3            1/4    <--> PB2          [probe]          Up
(vsh) show probe
 Name           Target
 ..................................
 PB2            RT3            1/4
```

Linking across servers with tunnels

A pair of virtual UDP tunnels connects two x86 servers—both of which are running vSLX—to form a larger virtual network.

NOTE
Creating virtual tunnels enables you to emulate larger virtual labs distributed over two or more servers.

1. On Server A, enter the **create tunnel** *tunnel-name peer-ip udp-port* command.

```
(vsh) create tunnel tu_serv_a_b 10.1.1.2 1000
```

2. On Server A, create a link between a virtual device to a "port" of the tunnel you just created.

```
(vsh) create link ch_9540_01 0/1 tu_serv_a_b 1/1
```

3. On Server B, enter the **create tunnel** *tunnel-name peer-ip udp-port* command.

```
(vsh) create tunnel tu_serv_b_a 10.1.1.1 1000
```

NOTE
Make sure that you specify the same *udp-port* as for the Server A tunnel.

4. On Server B, create a link between a virtual device to a "port" of the tunnel you just created.

```
(vsh) create link ch_9540_101 0/5 tu_serv_b_a 1/1
```

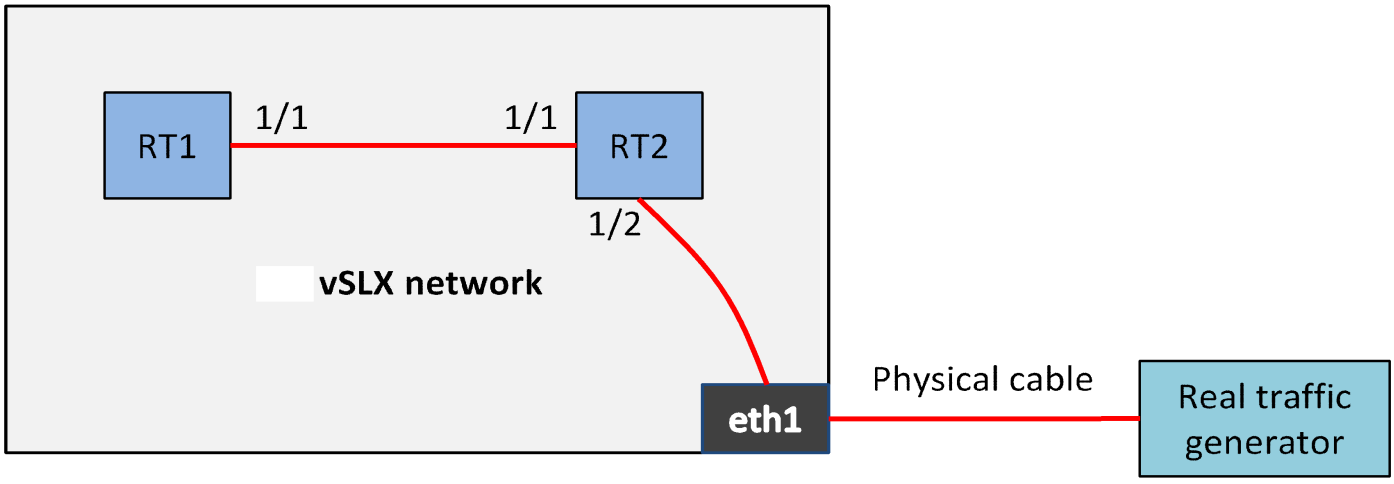
NOTE
Make sure that you specify the same tunnel *slot* **/** *port* as for the Server A link.

Linking to a real interface

Perform this task to link from a virtual device on a vSLX network to a real port on the server running vSLX.

> NOTE
> A common use-case is to support connection of an external traffic generator into a virtual network.

The following diagram displays a virtual network with two routers, RT1 and RT2. You link RT1 and RT2 and link RT2 to a real port (eth1) on the server hosting the vSLX instance. You can then connect an external device to that port.



1. Link RT1 and RT2.

```
(vsh) create link RT1 1/1 RT2 1/1
```

2. Link RT2 to the real interface eth1.

```
(vsh) create link RT2 1/2 -interface eth1
```

IP Fabric Emulation

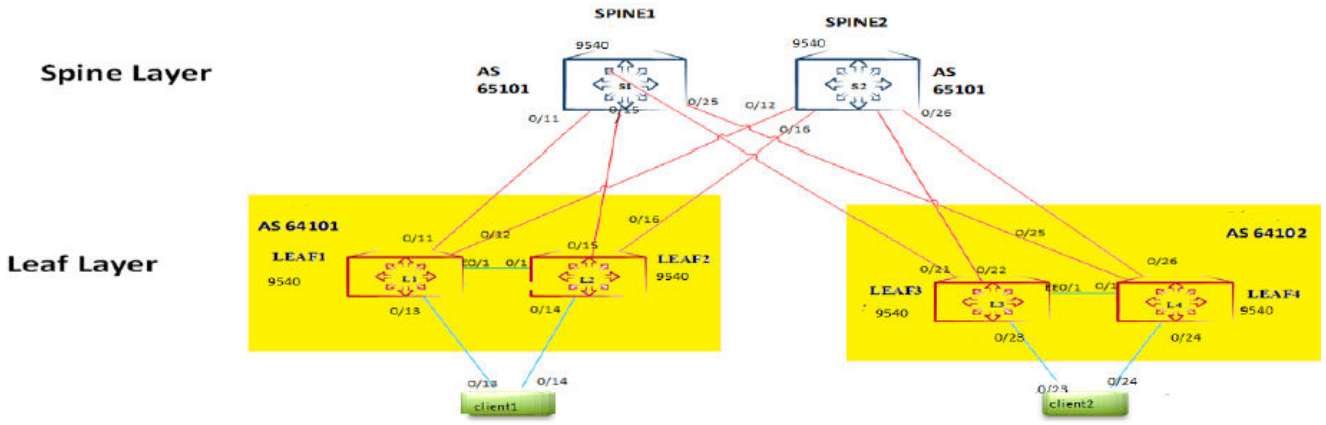
- IP Fabric reference topology 39
- IP Fabric features tested 39

IP Fabric reference topology

The following diagram illustrates a BGP-EVPN IP Fabric topology that you can emulate in vSLX.

NOTE
Although vSLX supports IP Fabric control-path functionality, data path is not yet supported.

FIGURE 1 BGP-EVPN IP Fabric reference topology



Hardware requirements:

- CPU: 24 cores
- Memory: 132 GB
- Storage: 512 GB

IP Fabric features tested

The following IP Fabric control-plane and data-plane features have been tested for this release.

IP control-plane features tested

The following IP Fabric control-plane features have been tested for this release.

- IP Fabric control-plane
- L2VNI (VLAN and bridge-domain)
- L3VNI (VLAN and bridge-domain)

IP data-plane features tested

The following IP Fabric data-plane features (VXLAN with ping on single VTEP) have been tested for this release.

- L2VNI (VLAN and bridge-domain)
- L3VNI (VLAN and bridge-domain)

Layer 2 Exchange Emulation

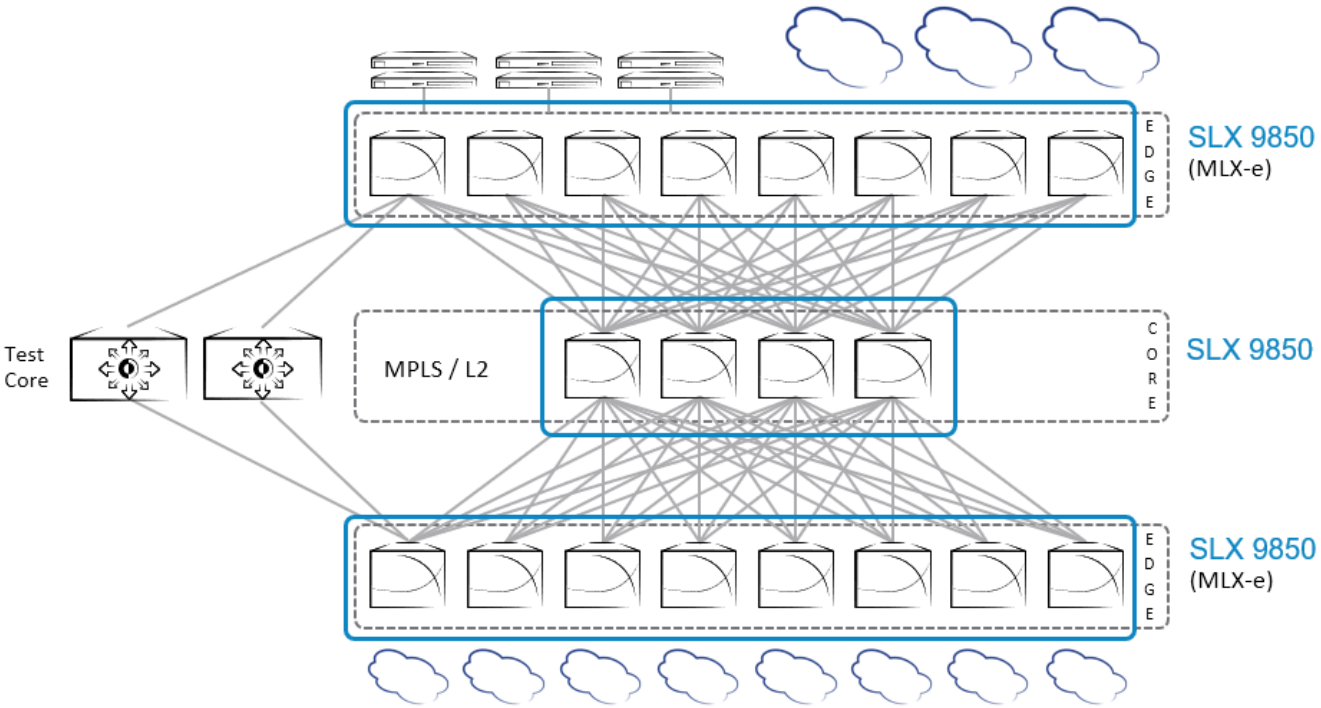
- L2 Exchange reference topology....41
- L2 Exchange features tested....42

L2 Exchange reference topology

The following diagram illustrates a Layer 2 Exchange topology that you can emulate in vSLX.

NOTE
Although vSLX supports control-path functionality, data path is not yet supported.

FIGURE 2 L2 Exchange reference topology



Key capabilities

The key capabilities supported are as follows:

- Layer 2—VLAN, bridge-domain (BD)
- Layer 3—Underlay with IPv4
- MPLS Control Plane—LDP, MPLS TE (RSVP-TE)
- VPLS and VLL Control Plane—LDP

L2 Exchange features tested

The following L2 Exchange features have been tested for this release.

VPLS features tested

The following Virtual Private LAN Service (VPLS) features have been tested for this release:

- VPLS PW bring-up with RSVP and OSPF as IGP
- Deletion and addition of MPLS config, bridge-domain with p2mp config, and tunnel config
- Verification of VPLS PW bring-up in raw ,tagged, and raw pass-through modes
- Verification of flapping IGP and RSVP sessions

VLL features tested

The following Virtual Leased Lines (VLL) features have been tested for this release:

- VLL PW bring-up with LDP and ISIS as IGP
- Deletion and addition of MPLS config, bridge-domain with p2p config, and tunnel config in VLL
- Verification of VLL PW bring-up in raw, tagged, and raw pass-through modes
- Verification of flapping IGP and LDP sessions

Known L2 Exchange issues

Ping is not currently supported under L2 Exchange.

Advanced vShell Features

- vShell commands from the Linux shell....43
- User-developed commands and scripts....43

vShell commands from the Linux shell

As an alternative to entering vShell commands from the vShell prompt, you can enter commands from the Linux shell.

The following Linux syntax is equivalent to `(vsh) show templates`:

```
root@your_user-Latitude-E5540:/home/your_user/lxc/vslxCont# vsh show templates
```

User-developed commands and scripts

Because vShell was developed using the Python programming language, you can extend vShell with Python, creating commands and scripts.

Python requirements for extending vShell

The guidelines for using Python to modify your vShell implementation are as follows:

- Although previous experience in Python programming is helpful, experience in other high-level languages is enough to get you started.
- To help decide which editor or integrated development environment (IDE) to use, refer to http://www.python.org.
- The supported version of Python is 2.7.

Implementation flow for vShell extensions

This topic presents a top-level view of how to add a command to your vShell instance.

1. Open a Python editor or integrated development environment (IDE) and create a new `*.py` file.
2. Type the required `from vsh import` statement.

```
# Import Python functions from the vsh module
from vsh import output, cmdparser
```

3. Create the target function.

```
# Define the function that the command will call.
def hello(username):

    output("Hello %s, how are you doing?", username)
```

4. Create the command that calls the target function.

```
# cmdparser(<target-function>, <syntax>, <msg>)
cmdparser(hello, "hello:%s", "Say hello to someone")
```

5. Save the file and copy it to the `~/.vsh` directory.

```
bash$ cp demo.py ~/.vsh
```

6. Start vShell.

```
$ vsh
For help anytime, type '?'
When you're done, type '^D'
(vsh)
```

7. Verify that the command you created is available.

```
(vsh) ?
 clear                  Clear
 connect                Connect console
 create                 Provisioning
 hello                  Say hello to someone
 delete                 Destroy
 exit                   Exit shell
 help                   Get help
 list                   Show running devices
 poweroff               Power off device
 poweron                Power on device
 show                   Show info
 start                  Power on and connect
 system                 Bridge daemon control
```

8. Enter the command—with required parameters—and verify that it performs as expected.

```
(vsh) hello Mark
 Hello Mark, how are you doing?
```

NOTE

The command you created will load every time you log in to this instance of vShell.

Syntax of the cmdparser( ) function

The `cmdparser(<target-function>, <syntax>, <msg>)` function creates commands that you can use in your vShell instance.

As indicated in Implementation flow for vShell extensions on page 43, `cmdparser( )` requires a target function.

<target-function>

Specifies the name of the function called by `cmdparser( )`.

<syntax> = "<vSLX-command-name>:<keyword>|<argument>:<keyword>|<argument>: ..."

Enclose the `<syntax>` element in quotes ("). The first word of the `<syntax>` element is the name of the vShell command you are creating, followed by the required keywords and arguments.

There is no special format for keywords. For each argument, you specify a type symbol, as follows:

TABLE 6 vShell data types

Data type	Type symbol
String	%s

TABLE 6 vShell data types (continued)

Data type	Type symbol
Integer	%d
IP address	%i
MAC address	%m or %M
Interface port	%I

For example, the syntax of the vShell `create link` command is `"create:link:%s:%I%s:%I"`. The elements of this command are as follows:

- Command name—`create`
- Keyword—`link`
- Arguments (separate by colons)—`%s:%I%s:%I`

Sample CLI input by vShell user:

```
create link rt1 0/1 rt2 0/1
```

<msg>

Specifies the CLI Help displayed when a user types `?`.

Enclose the `<msg>` element in quotes (").

```
"Creates a link between two entities."
```

Implementing a configuration script

This Python script creates the links in a network that includes a router, three workstations, a bridge, and a tunnel.

> **NOTE**
> Because vSLX configuration is persistent, this script is not needed for a single vSLX virtual lab. But it might be useful for setting up multiple networks to run multiple tests.

1. Open a Python editor or integrated development environment (IDE) and create a new `*.py` file.
2. Type the required `from vsh import` statement.

```
from vsh import m, cmdparser
# The function "m" defines a command executable in vShell.
```

3. Create the target function.

```
def demo():

    m("create link RT3 1/1 host6 1")
    m("create link RT3 1/2 -b BR1")
    m("create link RT3 1/5 T2 1")
    m("create link RT3 1/6 T2 2")
    m("create link -b BR1 host7 1")
    m("create link -b BR1 host30 1")
```

4. Create the `cmdparser(<target-function>, <syntax>, <msg>)` command that calls the target function.

```
cmdparser(demo, "demo_01", "Configures network of router, three workstations, bridge, and tunnel.")
# The demo target function does not require any parameters.
```

5. Save the file and copy it to the `~/.vsh` directory.

```
bash$ cp net_demo_01.py ~/.vsh
```

6. Start vShell.

```
$ vsh
For help anytime, type '?'
When you're done, type '^D'
(vsh)
```

7. Verify that the command you created is available.

```
(vsh) ?
 clear                  Clear
 connect                Connect console
 create                 Provisioning
 demo_01                Configures network of router, three workstations, bridge, and tunnel.
 delete                 Destroy
 exit                   Exit shell
 help                   Get help
 list                   Show running devices
 poweroff               Power off device
 poweron                Power on device
 show                   Show info
 start                  Power on and connect
 system                 Bridge daemon control
```

8. Enter the command that you created.

```
(vsh) demo_01
```

NOTE

The command you created will load every time you log in to this instance of vShell.

9. Enter the **show link** command to verify that your configuration was successful.

```
(vsh) show link
 Name           Port        Name           Port           State
 ..............................................................
 RT3            1/1    <--> host6          0/1            Up
 RT3            1/2    <--> BR1          [bridge]         Up
 RT3            1/5    <--> T2             0/1            Up
 RT3            1/6    <--> T2             0/2            Up
 host30         0/1    <--> BR1          [bridge]         Up
 host7          0/1    <--> BR1          [bridge]         Up
```

vShell Command Reference

- clear debug....47
- connect....49
- create bridge....51
- create chassis....53
- create link....55
- create probe....57
- create template....59
- create tunnel....61
- create workstation....63
- delete....64
- help....66
- list....67
- poweroff....68
- poweron....69
- show....70
- show debug....73
- start....74
- system....76
- vsh....77

clear debug

Clears all troubleshooting information and resets all counters to zero.

Syntax

clear debug

Command Default

Debug information is not deleted.

Modes

vShell prompt `(vsh)`

Usage Guidelines

There is no software limit to the debug cache.

This command does not have a **no** form.

Examples

The following example shows how to run the **clear debug** command.

```
(vsh) clear debug
(vsh)
```

History

Release version	Command history
17r.2.01	This command was introduced.

connect

Connects the current `vsh` session to the console of a virtual workstation, chassis, management module, or line card.

Syntax

connect *device-name* [**lc** *linecard-number* | **mm** *mm-number*]

Command Default

The console is not connected to any device.

Parameters

device-name
Specifies a virtual router, switch, or workstation.

lc *linecard-number*
(For SLX 9850) Specifies a line card. Supported values are 1 through 8.

mm *mm-number*
(For SLX 9850) Specifies a management module (MM). The only supported value is 1.

Modes

vShell prompt `(vsh)`

Usage Guidelines

Before you connect to an entity, you must turn it on, using the **poweron** command.

To turn on a device and connect to it with a single command, refer to start on page 74.

To disconnect the current `vsh` session from the device console, press **Ctrl-]**.

To log in to vShell and run this command, enter the `user@ubuntu:~$ vsh connect` command.

Examples

The following example shows how to connect the current `vsh` session to a device console.

```
(vsh) connect dev_01

SLX-OS (SLX)
SLX login:
```

The following example shows how to connect the current `vsh` session to a virtual SLX 9850 management module.

```
(vsh) connect dev_01 mm 1

SLX-OS (SLX)
SLX login:
```

The following example shows how to connect a console to a device line card.

```
(vsh) connect dev_01 lc 4

Ubuntu 14.04.5 LTS LCVM ttyS0

LCVM login:
```

The following example shows the error you get if you try to connect to a device that is turned off.

```
(vsh) connect h1
 h1: the device is powered off
```

History

Release version	Command history
17r.2.01	This command was introduced.

create bridge

Creates a virtual bridge to connect multiple ports and devices.

Syntax

create bridge *bridge-name*

Command Default

No bridges are defined.

Parameters

bridge-name

Specifies a unique name for the bridge. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

Modes

vShell prompt `(vsh)`

Usage Guidelines

A bridge forwards all traffic received by one port to all other entities connected to the bridge.

To connect as many entities as required to the bridge, use the **link bridge** command.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create bridge` command.

To delete a bridge, use the **delete bridge** command.

Examples

The following example shows how to create a bridge.

```
(vsh) create bridge bridge_01
```

The following example shows how to create a bridge, link two routers and a host through the bridge, display the links, and display the bridge summary.

```
(vsh) create bridge BR1
(vsh) create link -bridge BR1 RT1 4/10
(vsh) create link -bridge BR1 RT2 5/12
(vsh) create link -bridge BR1 host1 0/1
(vsh) show link
 Name           Port        Name           Port           State
 ..................................................................
 RT1            4/10   <--> BR1          [bridge]         Up
 RT2            5/12   <--> BR1          [bridge]         Up
 host1          0/1    <--> BR1          [bridge]         Up

(vsh) show bridge
 Name           No.connections
 ..................................
 BR1            3
```

History

Release version	Command history
17r.2.01	This command was introduced.

create chassis

Creates a bootable SLX 9540 or SLX 9850 virtual device from a vSLX template.

Syntax

create chassis *chassis-name* { **F4** | **F8** } *mm-cards slot* **/** *linecard-type template-name*

create chassis *chassis-name* **1U** *template-name*

Command Default

No chassis are defined.

Parameters

chassis-name

Specifies a unique name for the device. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

NOTE

Because chassis name-space is common to workstation name-space and tunnel name-space, we recommend naming conventions that prevent conflict.

F4

Specifies a virtual SLX 9850-4 device. Make sure that you enter "F4" and not "f4".

F8

Specifies a virtual SLX 9850-8 device. Make sure that you enter "F8" and not "f8".

1U

Specifies a virtual SLX 9540 device. Make sure that you enter "1U" and not "1u".

mm-cards

(Only for SLX 9850) Specifies the number of management module-cards. For this release, the only supported value is **1**.

slot **/** *linecard-type*

(Only for a SLX 9850 device) Specifies a slot and a line-card type. Separate multiple slot/card type specifications with commas and use a hyphen (-) to indicate a range of slots that share a common type of line card.

slot

Specifies the slot. For a SLX 9850-4 device, valid values are from 1 through 4 and you can specify as many as four cards. For a SLX 9850-8 device, valid values are from 1 through 8 and you can specify as many as eight cards.

linecard-type

Specifies the line-card type, which is either **72x10G** or **36x100G**.

template-name

Specifies the template.

Modes

vShell prompt `(vsh)`

Usage Guidelines

A chassis represents a virtual SLX 9540 or SLX 9850 device.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create chassis` command.

To delete a device, use the **delete chassis** command.

Examples

The following example shows how to create a virtual SLX 9850-4 device from the "9850_templ" template, defining slot 1 as 72x10G, slot 2 as 36x100G, and slot 4 as 72x10G.

```
(vsh) create chassis ch_9850_4_01 F4 1 1/72x10G,2/36x100G,4/72x10G 9850_templ
```

The following example shows how to create a virtual SLX 9850-4 device from the "9850_templ" template, defining slots 1 through 3 as 72x10G and slot 4 as 36x100G.

```
(vsh) create chassis ch_9850_4_02 F4 1 1-3/72x10G,4/36x100G 9850_templ
```

The following example shows how to create a virtual SLX 9850-8 device from the "9850_templ" template, defining all eight slots as 36x100G.

```
(vsh) create chassis ch_9850_8_01 F8 1 1-8/36x100G 9850_templ
```

The following example shows how to create a virtual SLX 9850-8 device from the "9850_templ" template, defining slots 1 through 3 as 72x10G, slot 4 as 36x100G, slots 5 through 7 as 72x10G, and slot 8 as 36x100G.

```
(vsh) create chassis ch_9850_8_02 F8 1 1-3/72x10G,4/36x100G,5-7/72x10G,8/36x100G 9850_templ
```

The following example shows how to create a virtual SLX 9540 device from the "9540_templ" template.

```
(vsh) create chassis ch_9540_01 1U 9540_templ
```

History

Release version	Command history
17r.2.01	This command was introduced.

create link

Emulates a cable connection for traffic between two entities.

Syntax

create link *first-entity second-entity*

create link *chassis-name slot* **/** *port second-entity*

create link *workstation-name slot* **/** *port second-entity*

create link *tunnel-name slot* **/** *port second-entity*

create link -bridge *bridge-name second-entity*

create link -interface *interface-name second-entity*

create link -probe *probe-name second-entity*

Command Default

No links exist.

Parameters

first-entity

Specifies the first device port, host-name, bridge, probe, or tunnel.

second-entity

Specifies the second device port, host-name, bridge, probe, or tunnel.

chassis-name

Specifies an SLX 9540 or SLX 9850 device for linking.

slot

Specifies a valid slot number. Must be 0 if the device does not contain slots.

port

Specifies a valid port number.

workstation-name

Specifies a workstation for linking.

slot

Specifies a valid slot number. Must be 0 if the device does not contain slots.

port

Specifies a valid port number.

tunnel-name

Specifies a tunnel interface for linking.

slot

Specifies a valid slot number. Values range from 0 through 15.

port

Specifies a valid port number. Values range from 1 through 4095.

-bridge *bridge-name*

Specifies a bridge for linking. You can abbreviate **-bridge** with **-b**.

-interface *interface-name*

Specifies a physical interface on the x86 server on which vSLX is running. You can abbreviate **-interface** with **-i**.

-probe *probe-name*

Specifies a probe for linking. You can abbreviate **-probe** with **-p**.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create link` command.

You can use the **create link -interface** *interface-name* option to connect a port on the x86 server to the virtual network you created. You can use this option to feed traffic-generator input into the virtual network.

In general, you link an entity from one server to an entity on another server through tunnel devices that are connected between the servers. (Tunnels are implemented in pairs between the two servers.)

To delete a link, use the **delete link** command.

Examples

The following example shows how to connect two ports on a virtual SLX 9540 with two ports on another SLX 9850.

```
(vsh) create link ch_9540_01 0/1 ch_9850_8_01 1/1
(vsh) create link ch_9540_01 0/3 ch_9850_8_01 5/1
```

The following example creates a link connecting port 1/1 on router RT1 to port 1/1 on router RT2 and a link connecting port 1/2 on router RT2 to an x86 server port with hostname "eth1".

```
(vsh) create link RT1 1/1 RT2 1/1
(vsh) create link RT2 1/2 -interface eth1 1
```

History

Release version	Command history
17r.2.01	This command was introduced.

create probe

Creates a virtual probe.

Syntax

create probe *probe-name*

Command Default

No probes are defined.

Parameters

probe-name

Specifies a unique name for the probe. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

Modes

vShell prompt `(vsh)`

Usage Guidelines

To connect a probe to an entity—a device port, a bridge, or a host interface—use the **link probe** command. Once the connection is created, traffic to and from the entity is forwarded to the probe.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create probe` command.

To delete a probe, use the **delete probe** command.

Examples

The following example shows how to create a probe, link it to a device port, display the links, and display the probe summary.

```
(vsh) create probe PB2
vsh) create link -probe PB1 RT3 1/4
(vsh) show link
 Name           Port        Name           Port           State
 ..............................................................
 RT1            1/2    <--> RT3            1/4            Up
 RT3            1/4    <--> PB2          [probe]          Up
(vsh) show probe
 Name           Target
 ..................................
 PB2            RT3            1/4
```

History

Release version	Command history
17r.2.01	This command was introduced.

create template

Creates a template for virtual SLX 9540 or SLX 9850 instances.

Syntax

create template *template-name* { **slx9540** | **slx9850** } *vslx-distribution-path*

Command Default

No templates are defined.

Parameters

template-name

Specifies a unique name for the template. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

slx9540

Specifies a template for virtual SLX 9540 instances.

slx9850

Specifies a template for virtual SLX 9850 instances.

vslx-distribution-path

Specifies the full path to the vSLX distribution installed on the device. The first character must be a slash (/). If the image is on a remote server, only NFS mount is supported.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create template` command.

To delete a template, use the **delete template** command.

Examples

The following example creates a SLX 9850 template.

```
(vsh) create template 9850_templ slx9850 /slxos-dist
```

History

Release version	Command history
17r.2.01	This command was introduced.

Release version	Command history
18r.1.00	The value of *vslx-distribution-path* was simplified.

create tunnel

A virtual tunnel connects two x86 servers—both of which are running vSLX—to form a larger virtual network.

Syntax

create tunnel *tunnel-name peer-ip udp-port*

Command Default

No tunnels exist.

Parameters

tunnel-name

Specifies a unique name for the tunnel. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

NOTE

Because tunnel name-space is common to workstation name-space and chassis name-space, we recommend naming conventions that prevent conflict.

peer-ip

Specifies the IP address of the other server.

udp-port

Specifies the port number, which must be identical on both servers and available on both servers.

Modes

vShell prompt `(vsh)`

Usage Guidelines

You need to create two tunnels, one on each server. A pair of tunnels can support thousands of virtual channels between the servers.

To delete a tunnel, use the **delete tunnel** command.

Examples

The following example uses a pair of virtual tunnels to connect virtual devices on two x86 servers—both of which are running vSLX—to form a larger virtual network.

```
(vsh) create tunnel tu_serv_a_b 10.1.1.2 1000
(vsh) create link ch_9540_01 0/1 tu_serv_a_b 1/1
(vsh) create tunnel tu_serv_b_a 10.1.1.1 1000
(vsh) create link ch_9540_101 0/5 tu_serv_b_a 1/1
```

History

Release version	Command history
17r.2.01	This command was introduced.

create workstation

Creates a virtual PC workstation.

Syntax

create workstation *workstation-name* **linux**

Command Default

No workstations are defined.

Parameters

workstation-name

Specifies a unique name for the virtual workstation. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

NOTE

Because workstation name-space is common to chassis name-space and tunnel name-space, we recommend naming conventions that prevent conflict.

linux

Specifies a workstation running Ubuntu Linux.

Modes

vShell prompt `(vsh)`

Usage Guidelines

The current version only supports virtual Ubuntu-Linux workstations.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create workstation` command.

To delete a workstation, use the **delete workstation** command.

Examples

The following example shows how to create a virtual Linux workstation.

```
(vsh) create workstation linux_ws_01 linux
```

History

Release version	Command history
17r.2.01	This command was introduced.
18r.1.00	The ubuntu.img guideline was deleted.

delete

Deletes virtual entities created under vSLX.

Syntax

delete { **bridge** *bridge-name* | **chassis** *chassis-name* | **probe** *probe-name* | **tunnel** *tunnel-name* | **workstation** *workstation-name* }

delete template *template-name* [**force**]

delete link -all

delete link device *entity-name*

delete link *entity-name slot* **/** *port* { **-bridge** *bridge-name* | **-interface** *interface-name* | **-probe** *probe-name* | *entity-name slot* **/** *port* }

delete link -bridge *bridge-name* { **-interface** *interface-name* | **-probe** *probe-name* | *entity-name slot* **/** *port* }

delete link -interface *interface-name* { **-bridge** *bridge-name* | **-probe** *probe-name* | *entity-name slot* **/** *port* }

delete link -probe *probe-name* { **-bridge** *bridge-name* | **-interface** *interface-name* | *entity-name slot* **/** *port* }

Parameters

bridge *bridge-name*
Specifies a bridge for deletion.

chassis *chassis-name*
Specifies an SLX 9540 or SLX 9850 for deletion.

probe *probe-name*
Specifies a probe for deletion.

tunnel *tunnel-name*
Specifies a tunnel for deletion. Deleting a tunnel also delete links connected to the tunnel.

workstation *workstation-name*
Specifies a workstation for deletion.

template *template-name*
Specifies a template for deletion.

force
Delete the template and all chassis (devices) created from it.

link
Specifies a link for deletion.

-all
Deletes all links in the current vShell instance.

device *entity-name*
Deletes all links between the specific chassis, workstation, or tunnel and all other entities.

entity-name slot **/** *port*
: Deletes a link between the specific chassis port, workstation, or tunnel and the following specified entity. For a workstation, the *slot* value is 0.

-bridge *bridge-name*
: Deletes links between the specific bridge and the additional specified entity.

-interface *interface-name*
: Deletes links between the specific physical interface on the vSLX x86 server and the following specified entity.

-probe *probe-name*
: Deletes links between the specific probe and the following specified entity.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh delete` command.

If you delete a device, all of the links connected to the device are deleted.

Examples

The following example shows how to delete a link between two virtual devices.

```
(vsh) delete link ch_9540_01 0/3 ch_9850_8_01 5/1
```

The following example shows how to delete all links.

```
(vsh) delete link –all
```

The following example shows how to delete a bridge and all links connected to it.

```
(vsh) delete bridge br_01
```

The following example shows how to delete a probe and all links connected to it.

```
(vsh) delete probe pr_01
```

The following example shows how to delete a tunnel and all links connected to it.

```
(vsh) delete tunnel tu_01
```

History

Release version	Command history
17r.2.01	This command was introduced.

help

Displays the vShell keyboard shortcuts.

Syntax

help

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and display the vShell keyboard shortcuts, enter the `user@ubuntu:~$ vsh help` command.

Examples

The following example displays the keyboard shortcuts.

```
(vsh) help
  ?  - for help
 Tab - auto fill
 ^P  - previous command
 ^N  - next command
 ^D  - end of the session
 ^A  - move cursor to the beginning
 ^E  - move cursor to the end
 ^W  - delete a word
 ^U  - delete from the beginning
 ^K  - delete to the end
```

The following example shows how to log in to vShell and immediately display the keyboard shortcuts.

```
user@ubuntu:~$ vsh help
  ?  - for help
 Tab - auto fill
 ^P  - previous command
 ^N  - next command
 ^D  - end of the session
 ^A  - move cursor to the beginning
 ^E  - move cursor to the end
 ^W  - delete a word
 ^U  - delete from the beginning
 ^K  - delete to the end
```

History

Release version	Command history
17r.2.01	This command was introduced.

list

Displays a list of virtual devices, specifying how many virtual machines (VM) are running on each device.

Syntax

list [**-all**]

Parameters

-all

Specifies to display all defined devices, including those with no VMs running.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To displays all defined devices—including those with no VMs running—include the **-all** keyword.

Examples

The following example displays all running devices.

```
(vsh) list
 Name                    No.running VMs
 ......................................
 a1                      1
 abc                     1
 abd                     1
 abe                     1
```

The following example displays all defined devices.

```
(vsh) list -all
 Name                    No.running VMs
 ......................................
 a1                      1
 a2                      0
 a3                      0
 abc                     1
 abd                     1
 abe                     1
```

History

Release version	Command history
17r.2.01	This command was introduced.

poweroff

Emulates turning off a router, switch, or workstation device.

Syntax

poweroff *device-name*

Command Default

The virtual status of all devices is turned off.

Parameters

device-name

Specifies the device name.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and initially run this command, enter the `user@ubuntu:~$ vsh poweroff` command.

Examples

The following example shows how to turn off a virtual device.

```
(vsh) poweroff ch_9850_4_02
```

History

Release version	Command history
17r.2.01	This command was introduced.

poweron

Emulates turning on a router, switch, or workstation device.

Syntax

poweron *device-name*

Command Default

The virtual status of all devices is turned off.

Parameters

device-name

Specifies the device name.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh poweron` command.

Examples

The following example show how to turn on a virtual device.

```
(vsh) poweron ch_9850_4_02
```

History

Release version	Command history
17r.2.01	This command was introduced.

show

Displays various aspects of the virtual system configured under vSLX.

Syntax

show chassis [*chassis-name*]

show link [**status**]

show { **bridge** | **probe** | **system** | **templates** | **tunnel** | **version** }

Parameters

chassis

Displays details of all chassis (routers or switches) or a specific chassis.

chassis-name

Displays details of a specific chassis.

link

Displays details of virtual cables between ports.

status

Also displays which sides of links are up or down.

bridge

Displays bridge connections.

probe

Displays probe names and targets.

system

Displays routers, switches, and workstations.

templates

Displays template names, types, corresponding SLX-OS versions, and the number of instances created.

tunnel

Displays tunnel names, peer IP addresses, ports, and statuses.

version

Displays the vShell version number.

Modes

vShell prompt `(vsh)`

Usage Guidelines

To log into vShell and run this command, enter one of the `user@ubuntu:~$ vsh show` commands.

Command Output

The **show system** command displays the following information:

Output field	Description
SLX 9540	Emulates an SLX 9540 switch.
SLX 9850	Emulates an SLX 9850 router.
VPC	Emulates a virtual PC running Linux.

Examples

To display details of all virtual routers and switches, enter the **show chassis** command.

```
(vsh) show chassis
 Chassis Name            Type            Power  Template
 ..........................................................................
 a1                      SLX 9540        on     Avalanche
 a2                      SLX 9540        on     Avalanche
```

To display details of all virtual routers, switches, and workstations, enter the **show system** command.

```
(vsh) show system
 Device Name             Type            Power
 ..................................................
 av1                     SLX 9540        off
 av2                     SLX 9540        off
 av3                     SLX 9540        off
 av4                     SLX 9540        off
 F1                      SLX 9850        off
 F2                      SLX 9850        off
 F3                      SLX 9850        on
 host1                   VPC             off
 host2                   VPC             off
```

To display details of virtual cables between ports, enter the **show link** command.

```
(vsh) show link
 Name           Port        Name           Port           State
 ..............................................................................
 a1             0/1    <--> a2             0/1            Down
 a1             0/5    <--> a2             0/5            Down
```

To display template names, types, corresponding SLX-OS versions, and the number of instances created, enter the **show templates** command.

```
(vsh) show templates
 Name                Type       SLXOS Version        Snapshots
 ..........................................................
 aaa                 slx9540    slx-os17r.2.01no     0
 ava                 slx9540    slx-os17r.2.01no     4
 abc                 slx9540    slx-os17r.2.01no     0
 router_1            slx9850    slx-os17r.2.01no     0
```

To display the vShell version, enter the **show version** command.

```
(vsh) show version
 VShell Version 1.6.3-0
 Copyright (c) Extreme Networks, Inc., 1996-2018
```

History

Release version	Command history
17r.2.01	This command was introduced.

show debug

Displays vSLX troubleshooting information.

Syntax

show debug

Modes

vShell prompt `(vsh)`

Usage Guidelines

Diagnostic commands are developed and intended for specialized troubleshooting. Please work closely with Extreme Networks technical support in running **debug** or **show system internal** commands and interpreting their results.

Examples

The following example displays a no-problem situation.

```
(vsh) show debug
Devices.....................................
Peers.......................................
a1[0.1] -- Int: Dev:
a2[0.1] -- Int: Dev:
a1[0.5] -- Int: Dev:
a2[0.5] -- Int: Dev:
Stats.......................................
Int a1[0.1] -- 0tx 0rx 0ttl
Int a2[0.1] -- 0tx 0rx 0ttl
Int a1[0.5] -- 0tx 0rx 0ttl
Int a2[0.5] -- 0tx 0rx 0ttl
Exception...................................
0load 1linkscan 0devput 0devbind 0ifput
0sock 0rcvloop 0errpkt 0errkey 0server
```

History

Release version	Command history
17r.2.01	This command was introduced.

start

Emulates connecting to a router, switch, or workstation device and turning it on.

Syntax

start *device-name*

Command Default

The virtual status of all devices is turned off.

Parameters

device-name

Specifies the device name.

Modes

vShell prompt `(vsh)`

Usage Guidelines

For the specific device, the **start** command implements the **poweron** command and then instantaneously the **connect** command. The output of the **start** command displays the messages generated as the connected device boots up.

To connect to a management module or linecard of an SLX 9850, refer to connect on page 49.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh start` command.

Examples

The following example displays typical output starting a switch or router.

```
(vsh) start RT1
 The device is powered on successfully (1)
Loading Linux 2.6.34.6 from partition 1 ...
[    0.000000] this is quiet!
[    0.000000] MTRR default type: write-back
[    0.000000] MTRR fixed ranges enabled:
[    0.000000]   00000-9FFFF write-back
[    0.000000]   A0000-BFFFF uncachable
[    0.000000]   C0000-FFFFF write-protect
[    0.000000] MTRR variable ranges enabled:
[    0.000000]   0 base 00C0000000 mask FFC0000000 uncachable
[    0.000000]   1 disabled
[    0.000000]   2 disabled
[    0.000000]   3 disabled
```

History

Release version	Command history
17r.2.01	This command was introduced.

system

Starts, stops, or restarts the vShell bridge daemon, which forwards traffic between linked ports.

Syntax

system { **start** | **stop** | **restart** }

Parameters

start

Starts the vShell bridge daemon.

stop

Stops the vShell bridge daemon.

restart

Stops and then starts the vShell bridge daemon.

Modes

vShell prompt `(vsh)`

Usage Guidelines

The vShell bridge daemon starts automatically with vShell. The **system** command is reserved for troubleshooting and development.

Examples

The following example stops the vShell bridge daemon.

```
(vsh) system stop
```

History

Release version	Command history
17r.2.01	This command was introduced.

vsh

Accesses the Extreme Virtual Shell (vShell), which enables you to create and manage virtual SLX-OS instances and networks.

Syntax

vsh

Modes

Ubuntu Linux on an x86 server

Usage Guidelines

If you apend a vShell command to `user@ubuntu:~$ vsh`, you access vShell and the command immediately runs.

After you are logged in to vShell, to display the top-level vShell commands, press **?**.

To exit vShell, enter **exit** or press **Ctrl-D**.

Examples

The following example shows how to access vShell from Ubuntu Linux.

```
user@ubuntu:~$ vsh
 For help anytime, type '?'
 When you're done, type '^D'
(vsh)
```

The following example shows how to display the top-level vShell commands.

```
(vsh) ?
 connect                Connect console
 create                 Provisioning
 delete                 Destroy
 exit                   Exit shell
 help                   Get help
 list                   Show running devices
 poweroff               Power off device
 poweron                Power on device
 show                   Show info
 start                  Power on and connect
```

The following example shows how to exit vShell back into Ubuntu Linux.

```
(vsh) exit
user@ubuntu:~$
```

History

Release version	Command history
17r.2.01	This command was introduced.

Appendix A: Alternate Routines

- Multiple vSLX labs (lxc)....79

Multiple vSLX labs (lxc)

(For the **lxc** flow only) Installing vSLX in Linux containers—rather than directly on the host—enables each user to have an independent virtual lab.

Linux Containers (LXC) support running multiple Linux systems on a control host using a single Linux kernel. We use *privileged* containers, created by root and running as root. After creating containers, you can install one vSLX virtual lab in each container.

Creating a container for vSLX (lxc)

(For the **lxc** flow only) This task is a prerequisite for installing each of the multiple instances of vSLX on the server.

> NOTE
> Although this legacy, unscripted flow is still supported, we recommend the scripted flow under Multiple vSLX labs on page 19.

1. Log in to the x86 server as a user with sudo privileges.

   > NOTE
   > Although all sudo users can create and access all containers, our user in this flow is `vlab1_user`, created in Creating additional Linux users on page 15.

2. Install `lxc`, using the following commands, confirming prompts to continue.

   ```
   vlab1_user@ubuntu:~$ sudo apt-get update
   vlab1_user@ubuntu:~$ sudo apt-get install lxc
   ```

3. Enter the **lxc-create** command to create the container.

   ```
   vlab1_user@ubuntu:~$ sudo lxc-create --name VLAB1 --template download --bdev dir -- --dist ubuntu --
   release xenial --arch amd64 --force-cache --no-validate --server images.linuxcontainers.org
   ```

   > NOTE
   > Because the previous example did not specify `-P <directory-path>`, the VLAB1 container is created in the default directory path, `/var/lib/lxc`. Command syntax is as follows:
   >
   > ```
   > sudo lxc-create [ -P <directory-path> ] --name <container-name> --template download --bdev
   > dir -- --dist ubuntu --release xenial --arch amd64 --force-cache --no-validate --server
   > images.linuxcontainers.org
   > ```

4. Verify container creation.

   ```
   vlab1_user@ubuntu:/slxos-dist$ sudo ls /var/lib/lxc
   ```

5. Create a `mkdev.sh` file and make it executable, as follows:
   a) Enter the **sudo vi** command.

```
vlab1_user@ubuntu:/slxos-dist$ sudo vi /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
```

   b) Copy the following content into `mkdev.sh`.

```
#!/bin/bash
#
# LXC autodev hook for Ubuntu 16 Container
#
# Some required device files for vSLX are
# not imported inside the container. So manually
# create them.
#
croot=${LXC_ROOTFS_MOUNT}
for i in `/usr/bin/seq 0 7`
do
    /bin/mknod $croot/dev/loop$i b 7 $i
done

mknod $croot/dev/loop-control c 10 237

for j in `/usr/bin/seq 0 11`
do
    /bin/mknod $croot/dev/dm-$j b 252 $j
done

/bin/mkdir -p $croot/dev/net
/bin/mknod $croot/dev/net/tun c 10 200

exit 0
```

   c) After you save and close the `mkdev.sh` file, make it executable.

```
vlab1_user@ubuntu:/slxos-dist$ sudo chmod +x /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
vlab1_user@ubuntu:/slxos-dist$ sudo ls -l /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
-rwxr-xr-x 1 root root 453 May  2 16:13 /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
```

6. Prepare the container `config` file, as follows:
   a) Enter the **sudo vi** command to open the file for editing.

```
vlab1_user@ubuntu:/slxos-dist$ sudo vi /var/lib/lxc/VLAB1/config
```

   The initial `config` file created by running **lxc-create** displays:

```
# Template used to create this container: /usr/share/lxc/templates/lxc-download
# Parameters passed to the template: --dist ubuntu --release xenial --arch amd64
# --force-cache --no-validate --server images.linuxcontainers.org
# Template script checksum (SHA-1): 9748088977ba845f625e45659f305a5395c2dc7b
# For additional config options, please look at lxc.container.conf(5)

# Uncomment the following line to support nesting containers:
#lxc.include = /usr/share/lxc/config/nesting.conf
# (Be aware this has security implications)

# Distribution configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs = /var/lib/lxc/VLAB1/rootfs
lxc.rootfs.backend = dir
lxc.utsname = VLAB1

# Network configuration
lxc.network.type = veth
lxc.network.link = lxcbr0  <-- Change it to br0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:b5:7d:b9
```

   b) Update the container `config` file with the values that you require. For example:

```
#  Template used to create this container: /usr/share/lxc/templates/lxc-download
# Parameters passed to the template: --dist ubuntu --release xenial --arch amd64
# --force-cache --no-validate --server images.linuxcontainers.org
# Template script checksum (SHA-1): 9748088977ba845f625e45659f305a5395c2dc7b
# For additional config options, please look at lxc.container.conf(5)

# Uncomment the following line to support nesting containers:
#lxc.include = /usr/share/lxc/config/nesting.conf
# (Be aware this has security implications)

# Distribution configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs = /var/lib/lxc/VLAB1/rootfs
lxc.rootfs.backend = dir
lxc.utsname = VLAB1

# Network configuration
lxc.network.type = veth
lxc.network.link = br0  <-- Changed from lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:b5:7d:b9

# vSLX configuration  <-- You need to add the folllowing sections:
lxc.aa_profile = unconfined
lxc.cgroup.devices.allow = c 10:236 rwm
lxc.cgroup.devices.allow = b 252:* rwm
lxc.cgroup.devices.allow = b 7:* rwm

# Expose tun device
lxc.cgroup.devices.allow = c 10:200 rwm

# To export SLX-OS distribution directory into the container. Change paths as needed.
```

```
# The following implementation uses container /slxos directory as the container
# mount point to access the SLX-OS software distribution.
# The host-specific path for the same directory is /var/lib/lxc/VLAB1/rootfs/slxos.
# This implementation mounts your_user's home directory to the mount point /slxos.
lxc.mount.entry = /slxos-dist /var/lib/lxc/VLAB1/rootfs/slxos-dist none bind 0 0
# syntax: /<SLX-OS-Build-dir>/slxos-dist <directory-path>/<container-name>/rootfs/slxos
# none bind 0 0

lxc.cgroup.devices.allow = c 10:237 rwm
lxc.autodev = 1
lxc.hook.autodev = ${LXC_ROOTFS_MOUNT}/root/mkdev.sh
```

c) Save and close the file.

7. In the container, create an `slx-dist` directory.

```
vlab1_user@ubuntu:/slxos-dist$ sudo mkdir /var/lib/lxc/VLAB1/rootfs/slxos-dist
```

Installing vSLX in a container (lxc)

(For the **lxc** flow only) After preparing a Linux container, perform this task to install a vSLX virtual lab in the container.

NOTE

Although this legacy, unscripted flow is still supported, we recommend the scripted flow under Multiple vSLX labs on page 19.

1. Start the container and attach to it.

```
vlab1_user@ubuntu:/slxos-dist$ sudo lxc-start -n VLAB1
vlab1_user@ubuntu:/slxos-dist$ sudo lxc-attach -n VLAB1
```

In these steps, the container name is "VLAB1".

2. Set a password for the default user "ubuntu".

```
root@VLAB1:/slxos-dist# passwd ubuntu
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@VLAB1:/slxos-dist# exit
exit
```

3. Log in to the container as "ubuntu".

```
user@ubuntu:/slxos-dist$ sudo lxc-console -n VLAB1
Connected to tty 1
Ubuntu 16.04.4 LTS VLAB1 pts/0

VLAB1 login: ubuntu
Password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-87-generic x86_64)

ubuntu@VLAB1:~$ sudo -i
[sudo] password for ubuntu:
root@VLAB1:~#
```

NOTE

To exit the container, type **Ctrl+a q**.

4. Become a root user.

```
ubuntu@VLAB1:~$ sudo -i
[sudo] password for ubuntu:
root@VLAB1:~#
```

5. Verify that all is ready for vSLX installation, as follows:
   a) List the `slxos-dist` directory.

```
root@VLAB1:~# ls /slxos-dist
extreme-lxc_2.0.0.deb  slxos17r.2.00  slxos17r.2.00.tar.gz  vslx_2.0.0.deb  vslx2.0.0.tar.gz
```

   NOTE
   You can also list included directories.

   b) Verify that the container's Ethernet interface is up and that an internet address is assigned.

```
root@VLAB1:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:16:3e:b5:7d:b9
          inet addr:192.0.2.0  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2001:db8::/64 Scope:Link
          inet6 addr: 2001:db8:ffff:ffff:ffff:ffff:ffff:ffff/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:59 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9877 (9.8 KB)  TX bytes:1312 (1.3 KB)
(output
                truncated)
```

   c) Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
root@VLAB1:~# ping example.com
```

6. Install the vSLX debian package.

```
root@VLAB1:~# dpkg -i /slxos-dist/vslx_2.0.0.deb
Selecting previously unselected package vslx-utils.
(Reading database ... 13031 files and directories currently installed.)
Preparing to unpack ./vslx_2.0.0.deb ...
Unpacking vslx (2.0.0) ...
Setting up vslx (2.0.0) ...
```

7. Run the post-installation script.

```
root@VLAB1:~# /VM/postinst-setup.sh
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Fetched 204 kB in 1s (202 kB/s)
Reading package lists... Done
Reading package lists... Done
```

After successful installation, proceed to Up-and-running example on page 28.